II Konferencja Naukowa KNWS'05
"Informatyka- sztuka czy rzemiosło"
15-18 czerwca 2005, Złotniki Lubańskie

ROZPROSZONY ALGORYTM WERYFIKACJI UKŁADÓW STEROWANIA

Agnieszka Węgrzyn

Instytut Informatyki i Elektroniki, Uniwersytet Zielonogórski
65-246 Zielona Góra, ul. Podgórna 50

e-mail: A.Wegrzyn@iie.uz.zgora.pl

STRESZCZENIE

Istnieje wiele metod weryfikacji układów sterowania z wykorzystaniem sieci Petriego. Jednakże metody te mają albo dużą złożoność obliczeniową, albo nie są zbyt dokładne. W prezentowanym podejściu przedstawiona zostanie dokładna metoda weryfikacji układów sterowania z wykorzystaniem hierarchicznych sieci Petriego. Metoda polega na wyznaczaniu blokad i pułapek w sieciach Petriego oraz sprawdzaniu zależności występujących pomiędzy nimi. W celu przyspieszenia otrzymywania wyników algorytm ten został rozproszony. Prezentowana metoda została przetestowana w środowisku klastra Linux.

1. WPROWADZENIE

Pomysł wykorzystania sieci Petriego w celu modelowania układów cyfrowych, a zwłaszcza kontrolerów logicznych, pojawił się pod koniec lat 70-tych. Aparat formalny sieci Petriego w połączeniu z metodami logiki formalnej umożliwia weryfikację i syntezę reprogramowanego sterownika logicznego metodą systematyczną. Układy współbieżne w intuicyjny sposób można zapisać w postaci interpretowanej sieci Petriego.

Specyfikacja funkcjonalna układu powstaje na podstawie analizy wymagań użytkownika. Zadaniem specyfikacji jest precyzyjne wyrażenie zewnętrznych skutków działania układów cyfrowych. Wykorzystując formalną specyfikację można już we wczesnym stadium projektowania wykryć i usunąć błędy. Specyfikacja formalna jednoznacznie określa postulowane działania układu cyfrowego.

Stosowanie sieci Petriego jako pośredniej formy specyfikacji pomiędzy opisem w języku naturalnym i specyfikacją logiczną ma wiele zalet. Do najważniejszych z nich należy możliwość weryfikacji opisanego algorytmu z wykorzystaniem bogatego aparatu matematycznego teorii sieci Petriego [1].

Weryfikację układu cyfrowego opisanego siecią Petriego można sprowadzić do badania pewnych własności sieci Petriego. Do najważniejszych z nich należą żywotność i ograniczoność. Cechy te zapewniają odpowiednio, że w układzie nie dojdzie do zapętlenia

procesów lub ich zatrzymania oraz, że dany proces wykonywany będzie skończoną liczbę razy.

Wyznaczanie wszystkich blokad i pułapek w sieci opisującej układ sterowania, pozwala na zlokalizowanie tych fragmentów sieci, w których występują defekty z punktu widzenia poprawnego przepływu sterowania w sterownikach logicznych.

2. ROZPROSZONY ALGORYTM WERYFIKACJI SIECI PETRIEGO

W rozdziale tym zostanie przedstawiony sposób optymalizacji algorytmu wyznaczania blokad i pułapek w sieci Petriego. Metoda wyznaczania blokad i pułapek bazuje na algorytmie Thelena, który wyznacza implikanty proste na podstawie danej funkcji boolowskiej.

2.1. Algorytm wyznaczania blokad i pułapek

W prezentowanym algorytmie topologiczna struktura sieci przedstawiona jest w postaci formuł Horna. W celu otrzymania rozwiązań, należy otrzymane równanie sprowadzić do postaci dysjunkcji. Rozwiązanie równania daje odpowiedź, czy sieć opisana danym równaniem spełnia określone własności, czyli w rozpatrywanym przypadku, czy zawiera blokady i pułapki. Rozwiązywanie formuł Horna polega na znalezieniu takich podstawień 0 lub 1 dla każdego literału z formuły, w taki sposób, że każda klauzula danej formuły będzie miała wartość 1, oraz formuła będzie miała wartość 1. Na podstawie wyrażeń przedstawionych w postaci koniunkcyjnej otrzymywane jest wyrażenie w postaci dysjunkcyjnej. W celu wyznaczenia rozwiązań dla formuł Horna, wykorzystywany jest algorytm Thelena-Mathonego. Algorytm ten jest symboliczną metodą otrzymywania implikantów prostych. Nie wykonywane są tutaj algebraiczne mnożenia, a tworzone i przeszukiwane jest jedynie drzewo, zgodnie z dobrze znanym algorytmem przeszukiwania grafu wgłąb – DFS. Opracowany algorytm polega na wyznaczaniu wszystkich blokad i pułapek dla podanej sieci Petriego. Ponieważ ich liczba może być bardzo duża, pierwszym zalecanym etapem algorytmu jest przygotowanie zredukowanej sieci Petriego, czyli usunięcie tych fragmentów sieci, które nie mają wpływu na jej żywotność. Następnie sieć Petriego przedstawiana jest w postaci dwóch formuł Horna (jedna opisująca blokady oraz druga opisująca pułapki). Kolejnym krokiem w proponowanym algorytmie jest przygotowanie drzewa Thelena wyznaczającego wszystkie rozwiązania dla podanej formuły Horna. Dokładny opis algorytmu został przedstawiony w [9]. Najbardziej czasochłonnym krokiem algorytmu jest tworzenie i przeszukiwanie drzewa, dlatego zdecydowano się na optymalizację właśnie tego etapu.

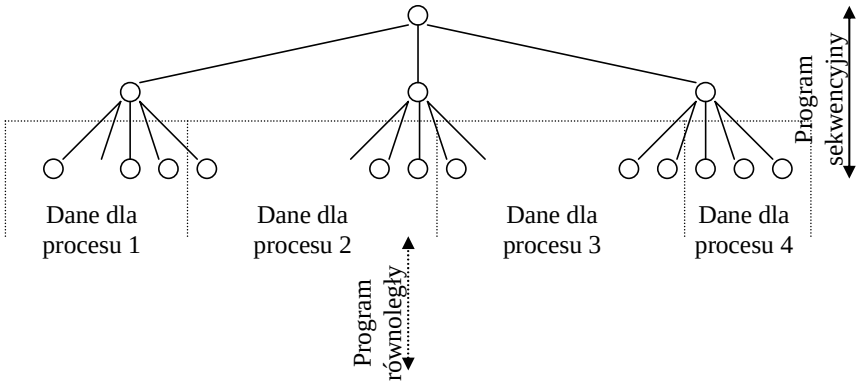
2.2. Algorytm wyznaczania implikantów prostych

Algorytm Thelena służy do wyznaczania implikantów prostych bazując na metodzie Nelsona, który udowodnił, iż można otrzymać wszystkie implikanty proste funkcji boolowskiej

w formie koniunkcyjnej przez jej przekształcanie do formy dysjunkcyjnej. Wykonanie przekształcenia za pomocą prostego mnożenia literałów jest czasochłonne i wymaga dużo pamięci na przechowywanie pośrednich wyników, których ilość zwiększa się w sposób wykładniczy. Metoda Thelena bazuje na drzewie przeszukiwań, a szybkość jej działania w znacznym stopniu zależy od posortowanych elementów funkcji. W drzewie przeszukiwań łuki reprezentują literały a wierzchołki przy łukach wymnożone literały od korzenia do danego wierzchołka. Wierzchołki tworzące liście nazywają się implikantami prostymi lub pochłanianymi. W strukturze drzewa implikanty pochłaniane nie są pamiętane dlatego też zmniejszyło się zapotrzebowaniem na pamięć. Dokładny opis sposobu generowania drzewa został podany w [6,8].

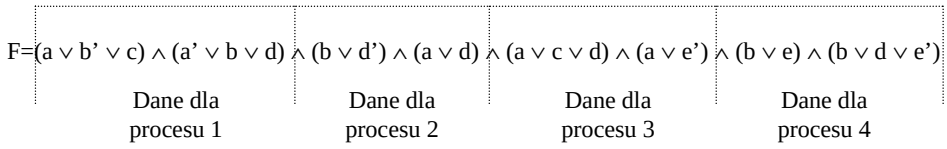
2.3. Modyfikacja algorytmu wyznaczania implikantów prostych

W celu optymalizacji algorytmu zaproponowano dwie metody podziału drzewa Thelena: metody poziomej i pionowej. Przy podziale pionowym procesy są generowane na podstawie częściowo wygenerowanego drzewa, dlatego konieczne było wstępne uruchomienie procesu optymalizacji (sekwencyjnie) do momentu uzyskaniu węzłów w drzewie w ilości równej lub przekraczającej ilość zadeklarowanych procesów (rys.1). W przypadku gdy ilość węzłów jest większa, procesy przeliczające lewą część drzewa przeszukiwań otrzymują większą ilość danych wejściowych z tego powodu, iż największy stopień optymalizacji otrzymuje się w tej właśnie części. Dzięki temu można w nieznaczny sposób wyrównać obciążenie procesów. Tak przygotowane dane mogły posłużyć do tworzenia pliku z danymi wejściowymi dla procesu. Pionowy podział drzewa tak jak i sekwencyjne generowanie drzewa przekształca funkcję z postaci koniunkcyjnej do postaci dysjunkcyjnej. Wyniki otrzymywane w obu przypadkach są jednakowe i niezależne od ilości generowanych procesów



Rys. 1. Podział danych z zastosowaniem metody pionowego podziału drzewa

W przypadku poziomego podziału drzewa wykorzystuje się jedną z właściwości funkcji matematycznych tzw. grupowanie klauzul (rys.2). Dlatego funkcję dzieli się na mniej więcej równe obszary i podaje się je procesom jako parametr wejściowy. Jednakże sam proces optymalizacji jest wykonywany w dwóch etapach tzn. przejście z postaci koniunkcyjnej na dysjunkcyjną i odwrotnie. W wyniku końcowym po połączeniu danych z poszczególnych procesów uzyskiwana jest postać koniunkcyjna.



Rys. 2. Podział danych z zastosowaniem metody poziomego podziału drzewa

3. TESTOWANIE APLIKACJI

Aplikacja została zaimplementowana z wykorzystaniem języka C. Procedura testowania została przeprowadzona w środowisku klastra MOSIX na platformie Linux.

3.1. Klaster

Klastry są obecnie najpopularniejszymi rozwiązaniami komputerów równoległych. Zakres ich stosowania rozciąga się od tradycyjnych stacji roboczych po komputery specjalnego przeznaczenia opartego na komputerach pracujących pod kontrolą systemu Linux.

Korzyści płynące z zastosowania klastrów:

- każdy komputer wchodzący w skład klastra może być używany jako serwer, czyli obok funkcji jaką spełnia w węźle klastra może pracować jako normalna stacja robocza,
- do obsługi klastra wystarczy tylko jedna konsola (koszty zakupu elementów klastra zmniejszone o zakup kart graficznych, klawiatur, monitorów),
- system łatwo się skaluje (klaster może składać się z kilku lub kilkudziesięciu elementów),
- lokalizacja uszkodzonego węzła jest bardzo proste i nie wymaga obecności przeszkolonego personelu. Heterogeniczna struktura sieci pozwala również wymienić uszkodzony element węzła na sprawny, mimo iż odbiega znacznie parametrami od oryginalnego.

Klastry posiadają natomiast znacznie gorsze parametry od komputerów SMP odnośnie komunikacji. Szerokość przenoszonego pasma jak i czas wyczekiwania dla łącz stosowanych w komputerach SMP wielokrotnie przewyższają parametry łącz klastrów [2].

W zaproponowanym przypadku komputer równoległy został oparty na architekturze klastra MOSIX [5]. Charakteryzuje się on bardzo prostą konfiguracją i stabilna pracą. Poza tym wkompilowany w jądro moduł MOSIX znakomicie zarządza, procesami w systemie rozproszonym, powodując dynamiczne przerzucanie procesów między węzłami klastra równoważąc ich obciążenie. Klaster MOSIX korzysta ze wszystkich zasobów systemu tzn. pamięci, interfejsów komunikacyjnych itp. Procesy zachodzące w systemie są całkowicie przeźroczyste dla programistów. Nie trzeba zabiegać o to by procesy na siłę umieszczać na określonym węźle monitorując przy tym jego stan. Całą kontrolę nad tym przejmuje system, a programista jedynie jest zobowiązany do generowania procesów i zainicjowania pomiędzy nimi komunikacji.

Klaster został zbudowanych na 4 komputerach klasy PC opartych na procesorach począwszy od Pentium II 400 MHz poprzez procesory AMD kończąc na Pentium IV 2,4 GHz. Różnorodność jednostek nie ma większego wpływu na pracę całego systemu, ale dzięki temu jest możliwość przetestowania takiej architektury. Wszystkie stacje robocze są sprzęgnięte za pomocą sieci Ethernet 100 Mb/s opartej na switchu. Struktura sieci i sam mechanizm klastra jest na tyle elastyczny że pozwala w dowolnej chwili włączyć dodatkowy komputer do sieci nawet podczas działającej aplikacji testowej. Systemem operacyjnym jest system Linux dystrybucja RedHat 8.0 i Mandrake 8.1. z wkompilowanym i skonfigurowanym jądrem MOSIX wersja 2.4-20. Klaster MOSIX również może pracować w interfejsie graficznym, co pozwoliło na generowanie ciekawych statystyk np. obciążenia procesora, zużycie pamięci RAM, użycia pliku wymiany (swap-u), obciążenia węzłów itp.

3.2. Uzyskane wyniki

Z przeprowadzonych badań wynika, że optymalizacja w znacznej mierze zależy od posortowania danych wejściowych. Dotyczy to zarówno metody pionowego jak i poziomego podziału drzewa przeszukiwań. Jednakże można zauważyć większą skuteczność działania drugiego algorytmu, który charakteryzuje się znacznie mniejszą ilością węzłów i implikantów nieprostych w wyniku końcowym, przy identycznych danych wejściowych. Większa skuteczność okupiona jest większym zapotrzebowaniem na moc obliczeniową, co wiąże się także z większym czasem wykonywania algorytmu.

Sekwencyjne testy szybkościowe wykonano na komputerze Pentium IV 2,4 GHz 256 MB RAM, natomiast środowisko rozproszone zostało zbudowane na czterech komputerach klasy PC pracujących pod systemem Linux z wkompilowanym w jądro systemu mechanizmem MOSIX. W skład klastra wchodziły komputery:

- Pentium IV 2,4 GHz, 256 MB RAM,
- AMD Athlon 1800 XP, 256 MB RAM,
- AMD Duron 600 MHz, 128 MB RAM,
- Pentium II 400 MHz, 192 MB RAM.

Pomimo, iż różnice w szybkości pracy węzłów klastra są ogromne, efektywność aplikacji pracującej na klastrze sięgała prawie 1500% (przyspieszenie 15-krotne). Podział danych na mniejsze porcje bardzo przyspiesza pracę całej aplikacji, gdyż poszczególne części drzewa są znacznie mniejsze i operacje na nim mogą się wykonywać dużo szybciej. Efektywność pracy aplikacji w systemie rozproszonym zwiększa się wraz ze wzrostem wielkości danych wejściowych. W przypadku małych rozmiarów danych wejściowych rozwiązanie sekwencyjne wykonuje się szybciej niż rozproszone, ponieważ podział danych dla kilku procesów zajmuje więcej czasu niż wykonanie całego algorytmu na jednym komputerze.

4. PODSUMOWANIE

Zaimplementowane metody podziału drzewa przeszukiwań w algorytmie Thelena mają wyraźny wpływ na czas pracy aplikacji, lecz uzyskanie większej skuteczności np. przy podziale poziomym drzewa przeszukiwań ma znacznie większe znaczenie.

Klaster MOSIX bardzo dobrze spełnia swoje zadanie polegające na zarządzaniu procesami w systemie rozproszonym. Efektywna realokacja procesów bardzo dobrze wpływa na efekt końcowy, którym jest czas wykonania aplikacji.

Praca naukowa (częściowo) finansowana ze środków Komitetu Badań Naukowych w latach 2004-2006 jako projekt badawczy (grant nr 3 T11C 046 26).

LITERATURA

[1] M. Adamski: *Projektowanie układów cyfrowych systematyczną metodą strukturalną*, Monografia Nr 49, Wydawnictwo Wyższej Szkoły Inżynierskiej w Zielonej Górze, Zielona Góra, 1990

[2] M. J. Bach: *Budowa systemu operacyjnego UNIX*, Wydawnictwo Naukowo Techniczne, Warszawa, 1995

[3] Z. Banaszak, J. Kuś, M. Adamski: Sieci Petriego. *Modelowanie, Sterowanie i Synteza Systemów Dyskretnych*, Wydawnictwo Wyższej Szkoły Inżynierskiej, Zielona Góra, 1993

[4] M. Ben-Ari: *Podstawy programowania współbieżnego i rozproszonego*, Wydawnictwo Naukowo Techniczne, Warszawa, 1996

[5] A. Kornacki, R. Gontarek: *Mosix cluster*, http://prioris.mini.pw.edu.pl /mosix/rk/rk.html

[6] H. J. Mathony: *Universal logic design algorithm and its application the synthesis of two-level switching circuits*, *IEE Proceedings*, Vol.136, Pt.E, No.3, 1989

[7] T. Murata: *Petri Nets: Properties, Analysis and Applications*, *Proceedings of the IEEE*, Vol.77, No.4, ss.541-580, April 1989

[8] B. Thelen: *Investigations of algorithms for computer-aided logic design of digital circuits*, (po niemiecku), PhD dissertation, ITIV, Univ. of Karlsruhe, 1988

[9] A. Węgrzyn: *Symboliczna analiza układów sterowania binarnego z wykorzystaniem wybranych metod analizy sieci Petriego*, Rozprawa doktorska, Politechnika Warszawska, Warszawa, 2002